Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий

Работа допущена к защите
Руководитель ОП
_____ А.В. Щукин
«_____» _____ 2020 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
МЕТОДИКА РАЗРАБОТКИ ВЫСОКОНАГРУЖЕННЫХ СИСТЕМ В ВЕБ-ПРИЛОЖЕНИЯХ

по направлению подготовки 09.04.03 Прикладная информатика
Направленность (профиль) 09.04.03_04 Прикладная информатика в области информационных ресурсов

Выполнил
студент гр. 3540903/80401 М.Е. Переверзев

Руководитель
доцент ВШИСиСТ,
к.т.н, А.В. Щукин

Консультант
ассистент ВШИСиСТ В.А. Пархоменко

Консультант
по нормоконтролю В.А. Пархоменко

Санкт-Петербург
2020

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий

УТВЕРЖДАЮ
Руководитель ОП
_____ А.В. Щукин
«_____» _____ 2020г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Переверзеву Максиму Евгеньевичу гр. 3540903/80401

1. Тема работы: Методика разработки высоконагруженных систем в веб-приложениях.
2. Срок сдачи студентом законченной работы: 20.05.2020.
3. Исходные данные по работе: _____
   3.1. Научные работы о подходах по построению высоконагруженных систем в веб-приложениях,
   3.2. Документация используемых фреймворков и инструментов, применяемых для разработки.
4. Содержание работы (перечень подлежащих разработке вопросов):
   4.1. Исследование источников определения высоконагруженных веб-приложений;
   4.2. Анализ источников по предметной области;
   4.3. Формулирование требований к высоконагруженным приложениям;
   4.4. Анализ имеющихся подходов и средств построения высоконагруженных веб-приложений;
   4.5. Выделение важных критериев высоконагруженных систем, разработка тестов для оценки;
   4.6. Оценка накопленных экспериментальных данных статистическими методами;

4.7. Проектирование и разработка веб-приложения в соответствии с результатами тестов.

5. Перечень графического материала (с указанием обязательных чертежей):
   5.1. Диаграмма классов.
6. Консультанты по работе:
   6.1. Ассистент ВШИСиСТ, В.А. Пархоменко (содержание и нормоконтроль).
7. Дата выдачи задания: 03.02.2020.

Руководитель ВКР _____ А.В. Щукин

Консультант _____ В.А. Пархоменко

Задание принял к исполнению 03.02.2020

Студент _____ М.Е. Переверзев

РЕФЕРАТ

На 43 с., 6 рисунков, 7 таблиц, 10 приложений.

КЛЮЧЕВЫЕ СЛОВА: СТИЛЕВОЕ ОФОРМЛЕНИЕ САЙТА, УПРАВЛЕНИЕ КОНТЕНТОМ, PHP, MYSQL, АРХИТЕКТУРА СИСТЕМЫ.

Тема выпускной квалификационной работы: «Методика разработки высоконагруженных систем в веб-приложениях».

В данной работе изложена сущность подхода к созданию динамического информационного портала на основе использования открытых технологий Apache, MySQL и PHP. Даны общие понятия и классификация IT-систем такого класса. Проведен анализ систем-прототипов. Изучена технология создания указанного класса информационных систем. Разработана конкретная программная реализация динамического информационного портала на примере портала выбранной тематики...

ABSTRACT

43 p., 6 figures, 7 tables, 10 appendices.

KEYWORDS: STYLE REGISTRATION, CONTENT MANAGEMENT, PHP, MYSQL, SYSTEM ARCHITECTURE.

The subject of the graduate qualification work is «The technique of developing highloaded systems in web applications».

In the given work the essence of the approach to creation of a dynamic information portal on the basis of use of open technologies Apache, MySQL and PHP is stated. The general concepts and classification of IT-systems of such class are given. The analysis of systems-prototypes is lead. The technology of creation of the specified class of information systems is investigated. Concrete program realization of a dynamic information portal on an example of a portal of the chosen subjects is developed...

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одним из явлений глобализации – это появления высоконагруженных систем (так же называемыми highload application / HLA). Социальные сети, банковские системы, обрабатывающие миллиарды транзакций в сутки, мессенджеры и другие системы существуют, чтобы удовлетворять потребность растущего населения. А такие приложения обусловлены высокой отказоустойчивостью и пропускной способностью.

Актуальность данной темы состоит в высоком спросе со стороны организаций, поддерживающие функциональность своего бизнеса, и для людей продвигающие свои стартапы, рассчитанные на широкий охват аудитории.

Новизна заключается в комплексности подхода, при разработке высоконагруженных систем в веб-приложениях. Подход обеспечивает разработку систем на основе количественных и качественных показателях, удовлетворяющих требованиям надёжности, предоставляя лучший выбор в соответствии с установленными требованиями.

Цель работы — формирование общей методики разработки высоконагруженных систем в веб-приложениях.

Для достижения поставленной цели, выделены следующие **задачи** дипломной работы:

1. Исследование источников определения высоконагруженных веб-приложений;
2. Анализ источников по предметной области;
3. Формулирование требований к высоконагруженным приложениям;
4. Анализ имеющихся подходов и средств построения высоконагруженных веб-приложений;
5. Выделение важных критериев высоконагруженных систем, разработка тестов для оценки;
6. Оценка накопленных экспериментальных данных статистическими методами;
7. Проектирование и разработка веб-приложения в соответствии с результатами тестов.

Методы исследования. Данная работа реализована с помощью метода научного исследования (design science research - DSR). Благодаря ему, производится чёткая идентификация и обоснование проблемы, определение целей, ограничений

и требований для проектирования и дальнейшей разработки артефакта – методика разработки высоконагруженных систем в веб-приложениях. Далее производится оценка артефакта со стороны эффективности и результативности, а так же других критериев. В конце, на его основе, разрабатывается веб-приложение, проходящее апробацию.

Для оценки экспериментальных данных используются методы математической статистики.

ГЛАВА 1. ВЫСОКОНАГРУЖЕННЫЕ ВЕБ-ПРИЛОЖЕНИЯ

1.1. Highload приложения

Современный человек уже не может представить свою жизнь без всевозможных сервисов, которые облегчают его проживание. А в свою очередь эти сервисы должны обеспечивать свои услуги всем желающим. Востребованность к определённым сервисам, среди пользователей, только увеличивается: мессенджеры, которые обрабатывают сообщения миллионов пользователей ежесекундно; банки, которые обслуживают потоки транзакций своих и не только клиентов; карты с актуальной информацией о загруженности трафика и так далее. Но так же появляются новые сервисы, предоставляющие свои услуги, которые так или иначе можно считать примерами highload приложения.

Так сложилось, что зачастую под высоконагруженной системой, люди представляют в основном веб-сервис в частности какой-нибудь многопользовательский интернет-сайт. Однако это лишь часть, к ним так же относятся всевозможные управляющие бизнес процессом и регулирующие системы.

Из-за специфики понятия highload сложно провести классификацию, так как нет такого понятия как «сайт/ сервис средней нагрузки». Каждый сервис/ сайт – специфичен. Для разных систем, равное количество запросов, приводит к разным нагрузкам на разные ресурсы. Однако любой сервис highload обладает одним или несколько критериев: [**souders2008high**]

- Большой поток единовременных пользователей;
- Наличие сложных многоуровневых/ многочисленных расчётов и вычислений;
- Большой объём обрабатываемых данных.

Традиционные качества highload систем: [**ge2009powerpack**]

- **Многопользовательность.** Высоконагруженные приложения направлены на потребителей. Следовательно в единицу времени они работают с более чем 1 человеком. Десятки, тысячи, сотни тысяч пользователей могут единовременно взаимодействовать, а предсказать верхний порог – невозможно. Однако заметим, что большое количество пользователей не гарантирует высокую нагрузку на саму систему.
- **Распределённость.** Зачастую высоконагруженные системы распределены на несколько узлов. Необходимость в распределении возникает в след-

ствии возрастающего объёма обрабатываемых данных или необходимости отказоустойчивости системы, чтобы приложение продолжало работать даже при отказе некоторых узлов. Заметим, что данные узлы могут располагаться как и на одной машине, когда создаются виртуальные образы узлов, так и на разных машинах.

– **Интерактивность.** В ключе высоконагруженных систем определяется способностью системы отвечать на входящие запросы пользователей за приемлемое время.

В качестве основных направлений высоконагруженных приложений можно выделить следующие: [**souders2008high**; **Nodejs82:online**]

– Мессенджеры. Один и самых распространённых примеров, где огромное число пользователей ежедневно общаются друг с другом, создавая лавину сообщений для сервиса. А так же на базе мессенджеров создаются целые корпоративные платформы для ведения бизнеса и не только. Наиболее часто упоминаемые представителями являются: Telegram, WhatsApp, Viber, Skype, Discord и так далее;
– Социальные сети. Аналогично как у с мессенджеров, но только разнообразие и количество контента больше чем у мессенджеров. Представителями являются: Facebook, Twitter, Instagram, Vk, Одноклассники и другие;
– Онлайн коммерция. Примеры различные интернет магазины такие как: Amazon, eBay и другие; или онлайн биржи;
– Образование. Образовательные площадки такие как Coursera, Edx И другие;
– Игровая индустрия. Сервера популярных площадок дистрибьюторов игр и предоставления серверов для онлайн игр. Примеры: Steam, Epic Games Store, Battle.net и другие;
– другие.

1.2. Основные уязвимости веб-приложений

Все веб-приложения подвержены стандартному набору угрозам [**wichers2017owasp**; **Securing91:online**], которые хоть и известны (достаточно редко появляется новый вид уязвимостей), но они постоянно усовершенствуются злоумышленниками. Атаки в обход аутентификации для не защищённых URL адресов, перехват данных администратора, инъекции в базе данных (SQL-инъ-

екции), cross site request forgery (CSRF), использование особенностей языков программирования, cross site scripting (XSS), небезопасное хранение важных данных и так далее.

Согласно открытому проекту обеспечения безопасности веб-приложений (Open Web Application Security Project - OWASP) за 2017 год определён следующий список уязвимостей для веб-проектах [**wichers2017owasp**]:

- Инъекции. Угроза внедрения, такие как SQL, NoSQL, OS и LDAP инъекции, возникают, когда посылаемые данные отправляются интерпретатору, в котором нет проверки и или экранирования потенциально опасных символов. Внедрённый код злоумышленника может заставить интерпретатора выполнить непреднамеренные команды или предоставить доступ к данным без надлежащей авторизации.
- Сломанная аутентификация. Фоновые функции, отвечающие за аутентификацию и управлением сеансами, часто реализуются неправильно, что позволяет злоумышленникам получить пароли, ключи или токены сеансов, а так же использовать другие недостатки реализации, чтобы временно или постоянно принимать идентификационные данные других пользователей.
- Конфиденциальные данные (Sensitive Data Exposure). Некоторые веб–приложения и API не защищают должным образом конфиденциальные данные. Злоумышленники могут перехватить их и модифицировать для совершения мошенничества с кредитными картами, кражи личной информации или других незаконных действий. Конфиденциальные данные могут быть скомпрометированы без дополнительной защиты, что требуют специальных мер предосторожности при взаимодействии с браузером.
- Внешние объекты XML (XXE). Многие старые или плохо настроенные процессоры XML анализирует внешние ссылки в документах XML. Таким образом сокрытый вредоносный код в XML исполняется, предоставляя злоумышленнику доступ или конфиденциальные данные.
- Broken Access Control. Организация ограничения на то, что разрешено делать аутентифицированным пользователям, часто не соблюдаются должным образом. Злоумышленники могут использовать эти недостатки для доступа к несанкционированным функциям и / или данным, таким как доступ к учетным записям других пользователей, просмотр конфиденциальных файлов, изменение данных других пользователей, изменение прав доступа и т. д.

- Неверная конфигурация безопасности (Security Misconfiguration). Неправильная настройка безопасности является часто встречающейся проблемой. Обычно это результат оставления конфигураций по умолчанию, открытого облачного хранилища, неправильно настроенных заголовков HTTP и не настроенный обработчик ошибок, выдающий конфиденциальную информацию при ошибках. Так же важно своевременное операционные системы, платформы, библиотек и приложения.
- Межсайтовый скриптинг XSS (Cross-Site Scripting). Внедрение вредоносного кода на выбранную страницу с целью взаимодействия внедрённого кода с сервером злоумышленников при открытии страницы.
- Небезопасная десериализация (insecure deserialization). Некоторые приложения восстанавливают данные из битовой последовательности (десерализация), тем самым создавая уязвимость. Злоумышленники могут подделывать эти последовательности, тем самым может изменить логику приложения, произведя удалённое выполнение кода, подделав структуру и объекты данных или же сфальсифицировать данные.
- Использование компонентов с известными уязвимостями. Компоненты, такие как библиотеки и другие программные модули, обладают и исполняются с теми же привилегиями, что и само приложение. Если в компоненте заложена уязвимость, то это предоставляет доступ к полному контролю сервера.
- Недостаточный сбор логов и мониторинга (Components With Known Vulnerabilities). Отсутствие грамотной системы логирования и мониторинга за приложением, не корректно формирует систему реагирования на инциденты. Таким образом, предоставляя злоумышленникам проводить «разведку» с дальнейшей атакой на систему.

В табл. 1.1 представлены методы борьбы против ранее перечисленных угроз.

На основе представленных угроз и способов борьбы с ними можно сформулировать общие требования к высоконагруженным сервисам:

- Спроектированная система должна иметь строгую архитектуру приложения;
- Потоки данных – определены;
- Использование библиотек и фреймворков предоставляют лучшую практику;
- Вводимые данные (пароли) должны храниться в зашифрованном виде;

Таблица 1.1

Список угроз с основными методами противодействия им

Угроза	Метод защиты
SQL injection	Формирование SQL запросов с экранированием потенциально опасных символов.
Сломанная аутентификация	Строгое формирование маршрутной карты совместно с допустимыми разделами приложения.
Sensitive Data Exposure	Хранение личных данных должно осуществляться строго в зашифрованном виде. Формирование системы доступа к данным по ключам.
XXE	Настроить процессор XML для использования локального статического Document Type Definition (DTD)[7] и запретить любое объявленное DTD, включенное в документ XML.
Broken Access Control	Запрещение доступа к функциональности по умолчанию. Использовать списки контроля доступа на основе ролей.
Security Misconfiguration	Отключение интерфейса администрирования и режим отладки приложения. Настройка сервера для предотвращения несанкционированного доступа, просмотра каталога и т. д.
Cross-Site Scripting	Фильтровать вводимые данные пользователя на основе ожидаемых. Использовать соответствующие заголовки ответа. Чтобы запретить XSS в ответах HTTP, которые не должны содержать HTML или JavaScript.
Insecure Deserialization	Не принимать сериализованные объекты из ненадежных источников или использовать среды сериализации, которые допускают только примитивные типы данных
Components With Known Vulnerabilities	Использовать специальные программы для контроля версий компонент и поиска уязвимостей.
Insufficient Logging & Monitoring	Подготовить план реагирования на инциденты. Реализовать систему создания журналов событий.

- Вся информация вводимая пользователем должна фильтроваться и экранироваться;
- Разграничение разделов приложения, согласно ролям и спроектированной маршрутизации;
- SQL-запросы, предварительно, должны быть подготовлены.

1.3. Выводы

В данной главе была произведена классификация highload приложений и они были определены. Были представлены направления, в которых используется высоконагруженные системы и примеры таких систем. Были проанализированы актуальные угрозы веб-приложений, а так же методы борьбы с ними.

ГЛАВА 2. АНАЛИЗ ИНСТРУМЕНТОВ И ПОДХОДОВ ПО СОЗДАНИЮ ВЫСОКОНАГРУЖЕННЫХ ВЕБ-ПРИОРЖЕНИЙ

2.1. Виды передачи данных

WebSockets - это тонкий транспортный уровень, построенный поверх стека TCP / IP устройства. Цель состоит в том, чтобы предоставить разработчикам веб-приложений то, что по сути является как можно более близким к исходному уровню связи TCP, при этом добавляя несколько абстракций, чтобы устранить некоторые разногласия, которые в противном случае могли бы существовать в отношении работы Интернета. Они также учитывают тот факт, что в Интернете существуют дополнительные соображения безопасности, которые необходимо учитывать для защиты как пользователей, так и серверов.

Long Polling - – это технология, которая реализует получение информации посредством «Длинных запросов». Механизм производи передачу данных следующим образом. Клиент запрашивает у сервера с помощью обычного http запроса. Сервер получает запрос и отправляет страницу. Данная страница при построении у клиента выполняет специальный скрипт, который отправляет запрос к серверу с указанием флага. Пока не обновится информация на сервере, данный запрос игнорируется. При появлении новой информации или её изменении сервер производит её рассылку. Клиент получает ответ с новой информацией и мгновенно

выполняется новый скрипт запроса к серверу, запуская новый процесс ожидания на нём.

Server-sent events (SSE) – эта технология похожа на WebSockets, однако в отличии от неё является односторонним каналом связи. Отправленные события передаются только с сервера на клиент и позволяют клиентам браузера получать поток событий от сервера через соединение HTTP без опроса. Клиент подписывается на «поток» с сервера, и сервер будет отправлять сообщения («поток событий») клиенту, пока сервер или клиент не закроет поток. Сервер сам определяет что и когда отправлять клиенту.

Ajax Polling – технология при которой клиент периодически отправляет серверу запросы XMLHttpRequest / Ajax через определенный интервал для проверки новых данных. Клиент инициирует запросы через небольшие регулярные интервалы (например, 0,5 секунды) Сервер готовит ответ и отправляет его клиенту, как обычные HTTP-запросы. Повторные запросы к серверу тратят ресурсы, так как каждое новое входящее соединение должно быть установлено, HTTP-заголовки должны быть переданы, должен быть выполнен запрос новых данных, а также должен быть сгенерирован и доставлен ответ (обычно без новых данных). Соединение должно быть закрыто и все ресурсы очищены.

Forever Frames – создает скрытый iframe (тэг), который выполняет запрос к конечной точке на сервере, который не завершается. Затем сервер непрерывно отправляет клиенту сценарий, который немедленно выполняется, обеспечивая одностороннее соединение в реальном времени между сервером и клиентом. Соединение от клиента к серверу использует отдельное соединение от сервера к клиентскому соединению, и, как и стандартный HTTP-запрос, новое соединение создается для каждого фрагмента данных, который необходимо отправить.

В табл. 2.1 приведены виды транспортировки данных, которые чаще используются.

Наилучшим видом передачи данных является WebSockets. Выбор производился с помощью метода Саати (реализация метода Саати Приложение 1).Расчёты можно посмотреть в Приложение 2. Из этого следует, что такой подход предоставляет лучшее решение из представленных вариантов, так как использует меньший объём служебной информации, является двунаправленным, полнодуплексным соединением и для взаимодействия поддерживает одно соединение.

Таблица 2.1

Сравнение видов передачи данных [**WebSocke87:online**; **LongPoll65:online**; **Pollingv75:online**]

	WebSockets	Server-sent events (SSE)	Long Polling	Ajax Polling
Количество запросов на опрос сервера	1 на опрос сервера (установление соединения)	1 на опрос сервера (установление соединения)	1 на опрос сервера	1 >= на опрос сервера
Количество служебных заголовков вместе с ответом	1 заголовок в момент подключения клиента к серверу	1 заголовок в момент подключения клиента к серверу	1 заголовок за опрос сервера	1 заголовок за опрос сервера
Служебная информация прикрепляемая в ответе и запросе	2-10 байт у исходящих, в зависимости от полезной нагрузки	7-18 байт у исходящие	200 байт у запроса и ответа	200 байт у запроса и ответа

2.2. Инструменты разработки высоконагруженных приложений

2.2.1. Введение

Разработка любого приложения начинается с разработки архитектуры. Архитектура – это фундамент приложения, который оказывает большое влияние на конечный продукт, в особенности для highload приложения, так как отвечает за успешность и жизнеспособность системы.

Спроектировав архитектуру приложения, надо определить его ядро. Ядро обеспечивает связь между веб-клиентами и сервером. Множество языков программирования обладают возможностями для осуществления такого взаимодействия. Кроме того к некоторым из них прилагаются специализированные библиотеки и фреймворки, упрощающие как процесс разработки, так и улучающие базовый функционал языков. Выбор языка зависит от вида задачи, которую необходимо решить или от требований заказчика. Для высоконагруженных систем условно можно выделить 2 вида:

– CPU bound;

Рис.2.1. Инструменты разработки приложения

– I/O bound.

CPU bound (аппаратные ограничения) – задачи связанные с высокой вычислительной сложностью, такие как: специфичные алгоритмы, некоторая математика, задачи кодирования и шифрования и так далее. I/O bound (ограничения на ввод/вывод) – задачи в которых идут много операции с вводом выводов. Определив вид основной задачи проектируемого приложения определяем ключевые критерии выбора конкретного языка программирования по средством их сравнения (сравнение языков программирования производится точно так же как и в главе «2.1 Виды передачи данных»).

Из основных подходов реализации можно выделить:

– Разработку приложения средствами самого языка программирования

– Разработку приложения совместно с фреймворками и внешних библиотек.

На Рис. 2.1 представлены существующие инструменты для разработки веб-приложений реального времени.

2.2.2. Языки программирования

Выбор правильного языка программирования имеет решающее значение в разработке приложения. Проведём исследование языков программирования, подходящих для серверной веб-разработки, таких как Python, Node.js, Java, PHP. Языки сравниваются по общей популярности, поддержке облачных провайдеров, производительности и доступным интеграциям.

Python – это интерпретируемый объектно-ориентированный язык программирования высокого уровня с динамической семантикой. Его встроенные структу-

ры данных высокого уровня в сочетании с динамической типизацией и динамическим связыванием делают его очень привлекательным для быстрой разработки приложений, а также для использования в качестве скриптового или связующего языка для соединения существующих компонентов. Python поддерживает модули и пакеты, что способствует модульности программы и повторному использованию кода. Интерпретатор Python и обширная стандартная библиотека доступны для всех основных платформ.

Node.js – это среда, которая позволяет использовать JavaScript как для backend, так и для frontend разработки, а также для решения проблем совместимости. Его также можно определить как язык сценариев на стороне сервера. Он хорош для обработки проектов с множеством одновременных подключений или приложений с высокоскоростным и интенсивным вводом / выводом (I/O), а также приложений, таких как платформы повышенной производительности (например, системы управления контентом), торговые площадки P2P и платформы электронной коммерции.

Java – это объектно-ориентированный и параллельный язык программирования общего назначения, разработанный Sun Microsystems в 1995 году. Он использует механизм под названием JVM (виртуальная машина Java), который обеспечивает среду выполнения для запуска кода Java и его приложений. Он переводит байт-код Java в язык, который может интерпретироваться машинами. JVM является частью JRE (Java Runtime Environment).

PHP – это язык программирования, созданный для создания веб-приложений, построенный на языке программирования C и использующий уникальные HTML-подобные теги для содержания своего кода. Язык программирования PHP в основном используется на стороне сервера, что означает, что он работает на программном обеспечении вашего веб-сервера, которое обычно будет обслуживать HTML ваших посетителей.

Наилучшим языком программирования, согласно представленным критериям и по расчётам метода Саати – является Java (Приложение 1). Из этого следует, что такой подход предоставляет лучшее решение из представленных вариантов, однако в разработке для сервера будет использоваться Node js (согласно требованиям заказчика).

Таблица 2.2

Сравнение языков программирования [**PHPvsPyt48:online**; **NodejsVs38:online**] Приложение [**appendix:comparationPL**]

	Python	Node Js	Java	PHP
Наличие библиотек	Обладает исключительно хорошо развитой библиотечной поддержкой практически для всех типов приложений	Обладает исключительно хорошо развитой библиотечной поддержкой практически для всех типов приложений	Hibernate, Swing, SWT (Standard Widget Toolkit)	Packagist (репозиторий пакетов PHP) является сильной основой, поддерживающей PHP
Доступ к базам данных	Не сильная интеграция базы данных	Он обеспечивает доступ к более чем 20 различным базам данных	Он обеспечивает доступ к более чем 20 различным базам данных	Он обеспечивает доступ к более чем 20 различным базам данных
Скорость работы	4/5	5/5	5/5	4/5
Наличие фреймворков	Django, Flask, Pylons, Pyramid	Express.js, Meteor.js, Koa.js	Java Server Faces (JSF), Stuts2, Spring	Codeigniter, Zend, Laravel, Symfony

2.2.3. Библиотеки

Чтобы обеспечить двустороннюю связь клиента с сервером выделим решения, считающиеся популярными, для работы с web sockets и Node.js.

Socket.io – это библиотека позволяет установить двунаправленную связь между клиентом и сервером. Для корректной работы библиотеки, ей необходимо установить как на клиентской стороне, так и на серверной Node js. API у компонент (клиентская, серверная) – идентичен и управляет событиями. Socket.IO напоминает WebSockets. WebSockets также является браузерной реализацией, позволяющей осуществлять двунаправленную связь, однако Socket.IO не использует это в качестве стандарта. Во-первых, Socket.IO создает соединение с длинным опросом, используя xhr-опрос. Затем, как только это установлено, он обновляется

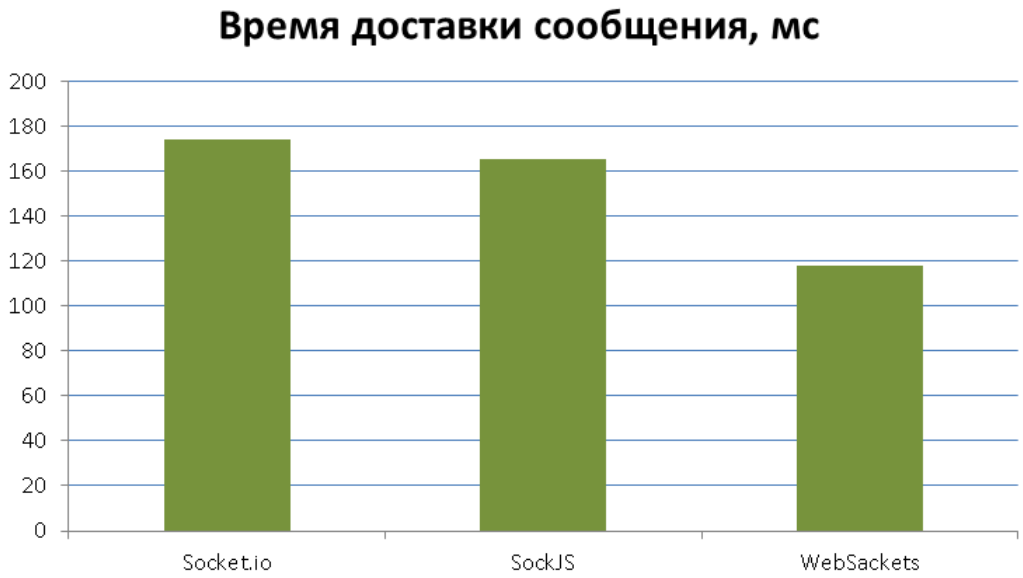

Рис.2.2. Сравнение времени опроса

до наилучшего способа подключения. В большинстве случаев это приведет к соединению WebSocket. Хотя Socket.IO может использоваться в качестве оболочки для WebSocket так как предоставляет больший функционал, включая широковещательную передачу на несколько сокетов, асинхронный ввод-вывод и хранение данных.

Sock.js - то JavaScript-библиотека браузера, которая предоставляет объект, подобный WebSocket. SockJS предоставляет вам согласованный кросс-браузерный API Javascript, который создает полнодуплексный междоменный канал связи с низкой задержкой между браузером и веб-сервером.

Произведём сравнение WebSocket [**WebSocke16:online**] с популярными библиотеками [**Introduc57:online**; **SockJSBчG77:online**], работающими на этом протоколе, однако предлагающими больший функционал. Для сравнения были разработаны клиенты и серверы (Приложения 5, 6), используя express.js (фреймворк для создания сервера Node.JS).

2.2.4. Мониторинг

Так как веб-серверы обрабатывают запросы пользователей на контент, то это непосредственно отражается на производительности и заметное влияние на взаимодействие с пользователя. Если веб-серверы работают медленно, пользователи откажутся от его услуг. Поэтому Application Performance Monitoring (APM) необходим для проекта. Он предупредит вас о любых ошибках или сбоях, которые могут привести к простою и о возможных сбоях.

Одним из основных преимуществ мониторинга – автоматизация. Сайты высокой ежедневной проходимостью часто оптимизируют пропускную способность балансировкой нагрузки, когда запросы делегируются на несколько веб-серверов. Отдельная служба балансировки нагрузки принимает входящие запросы, проверяет доступность веб-серверов, расположенных за ней, и передает запрос на доступный сервер. Для этого балансировщик нагрузки должен знать текущую нагрузку каждого веб-сервера и его доступность для обработки новых запросов.

Наконец, мониторинг помогает отслеживать популярность и рост веб-сайтов и веб-приложений. Метрики трафика и подключений позволяют получить непосредственное представление об активности сайта, в том числе о количестве активных пользователей и продолжительности каждого сеанса. Эти данные могут помочь вам разработать планы по масштабированию вашего веб-сайта, оптимизации вашего приложения или развертыванию других служб для поддержки возросшего спроса.

New Relic – является одним из самых популярных инструментов в среде APM. Он предоставляет информацию о многих типах программных и аппаратных компонентов или о взаимодействии с пользователем и поддерживает множество технологий, либо через своих собственных агентов, либо через внешние плагины. Одной из лучших функций является подробный мониторинг транзакций, обеспечивающий отслеживание между приложениями и отслеживание производительности операторов SQL. [**NewRelic78:online**]

Преимущества:

- Дружественный пользовательский интерфейс;
- Очень хорошо в нахождении корреляции данных;
- Включает отслеживание транзакций и процессов;
- Хороший форум сообщества и поддержка.

Недостатки:

- Предоставляется только по SaaS (нет локальной версии);
- Хранение данных зависит от уровня подписки, но в лучшем случае сохраняет все данные в течение 8 дней (после сохраняет только усреднённые данные);
- Ценообразование. Стоимость подписки начинается примерно с 60 долларов США за месяц и увеличивается в зависимости от многих аспектов (уровня подписки, компонентов мониторинга, количества серверов и т. д.). С другой стороны, он имеет бесплатную версию Lite, но она сильно

урезана и имеет только 2 часа хранения ограниченного количества данных (строго определённых).

CloudWatch – это инструмент мониторинга и управления Amazon Web Service (AWS). Он предоставляет данные о производительности компонентов AWS и приложений, работающих в инфраструктуре Amazon. CloudWatch имеет базовый мониторинг, который является бесплатным для пользователей AWS, а так же подобрав тарифный план его можно улучшить. Предоставляя такие функции, как более подробный мониторинг, пользовательские метрики и многое другое. Однако пользовательский интерфейс может быть немного сложным для людей, не имеющих опыта работы с подобными инструментами.[**AmazonCl81:online**]

Преимущества:

- Бесплатная версия (для пользователей AWS) предлагает достаточно ресурсов для базового мониторинга приложения;
- Настраиваемые панели для отображения метрик и графиков;
- Гибкие низкие цены.

Недостатки:

- Он может использоваться только для компонентов AWS и, следовательно, для приложений, работающих только на серверах Amazon. Есть несколько скриптов, созданных сторонними разработчиками, для получения метрик для серверов не-AWS, но они не являются «официальным»;
- Пользовательский интерфейс не очень удобен для анализа данных, что затрудняет выполнение корреляции данных;
- Нет отслеживания транзакций;
- По умолчанию нет метрик использования памяти. Пользовательский показатель должен быть настроен для мониторинга этого основного индикатора.

Dynatrace APM – предлагает мониторинг производительности и управление приложениями как в модели SaaS, так и на локальном сервере. Он поддерживает несколько технологий, имеет глубокую трассировку транзакций, мониторинг взаимодействия с пользователем (синтетический мониторинг и мониторинг реального пользователя) и мониторинг сети. Так присутствует функция, которая автоматически обнаруживает все компоненты сервера и быстро запускает отчетные данные на разных уровнях после его установки. Даже время загрузки страницы сообщается путем внедрения JavaScript в ваше веб-приложение.[**Applicat35:online**]

Преимущества:

– Простота установки и настройки;
– Бесплатная версия позволяет контролировать до 5 серверов в течение неограниченного времени, без ограничений функциональности или срока хранения данных. Там количество посещений ограничено 100к;
– Обнаруживает топологию приложений, развертывания и изменения среды в режиме реального времени;
– Отлично подходит для отслеживания транзакций и процессов.

Недостатки:

– Пользовательский интерфейс немного сложен, особенно для пользователей, не имеющих опыта работы с подобными инструментами;
– Ценообразование. Несмотря на то, что бесплатная версия действительно хороша для небольших инфраструктур (5 серверов или меньше), платные опции имеют цены, аналогичные New Relic.

AppDynamics – несколько лет назад этот инструмент был основным современным программным обеспечением для мониторинга APM. Рассматривая некоторые функции AppDynamics (например, главный экран с картой служб, используемых с их нагрузкой на вызов и индексом работоспособности, или мониторами взаимодействия с пользователем), создается впечатление, что приоритетами мониторинга инструментов являются крупные предприятия с большой инфраструктурой.[**Theworld87:online**]

Преимущества:

– Дружественный пользовательский интерфейс с настраиваемыми инструментальными панелями;
– Хороший мониторинг ресурсов сервера;
– Включает транзакции и отслеживание процессов;
– Использует алгоритмы машинного обучения, чтобы установить базовые значения для установки оповещения.

Недостатки:

– Некоторые функции инструмента основаны на Flash, и вы не можете использовать их, не включив его в браузере;
– Ценообразование. AppDynamics нет цен, опубликованных на веб-странице, вам придется связаться с отделом продаж, чтобы получить индивидуальный план и стоимость, основанные на ваших потребностях.

CA Technologies APM – одной из основных характеристик инструмента CA APM является то, что это гибкое и настраиваемое программное обеспечение:

компания предлагает службу поддержки, которая позволяет настраивать особый мониторинг и оповещения для получения более точных и полезных данных для каждого специфическая для компании ситуация.

CA APM имеет настраиваемые панели мониторинга, многоперспективные представления о работе конечных пользователей, автоматические трассировки транзакций и множество показателей ресурсов инфраструктуры.[**Applicat77:online**]

Преимущества:

- Универсальный инструмент, так как компания CA готова адаптировать инструмент для нужд клиента;
- Team Center предоставляет настраиваемую панель инструментов для быстрой навигации по деталям, чтобы разобраться в проблемах;
- Включает отслеживание транзакций и процессов;
- Хорошая поддержка и форум сообщества.

Недостатки:

- Это может потребовать довольно большой кривой обучения, особенно для людей, не имеющих опыта работы с такими инструментами;
- Цены могут быть немного высокими для небольших инфраструктур.

Таблица 2.3

Сравнение APM

	User Interface	SaaS/ Local server	Transaction trecking	Data correlation	Support services
New Relic	Простой и понятный интерфейс	SaaS	Подробный мониторинг транзакций	Хорошо реализован	Имеется
Cloud Watch	Требует определённые навыки работы в данной сфере	SaaS и локальный сервер (только для AWS)	Отсутствует	Имеется, слабая реализация	Имеется, но только для клиентов AWS
Dynatrace APM	В начале кажется сложным, но к нему быстро привыкнуть	SaaS и локальный сервер	Отличное (подробное) отслеживание проходящих транзакций и процессов	Хорошо реализован	Имеется, но уровень помощи зависит от подписки
App Dynamics	Простой и понятный интерфейс, только используется Flash	SaaS и локальный сервер	Подробный мониторинг транзакций	Имеется	Имеется
CA APM	Простой и понятный интерфейс	SaaS и локальный сервер	Подробный мониторинг транзакций	Хорошо реализован	Имеется

Наилучшей системой мониторинга, согласно представленным критериям и по расчётам метода Саати – является Dynatrace APM (Приложение 7). Из этого следует, что такой подход предоставляет лучшее решение мониторинга из выше представленных.

2.2.5. Веб-хостинг

2.2.5.1. Виды хостингов

Выбирать хостинг, на котором будет размещаться высокопроизводительное веб-приложение – это ещё один важный пункт. Если все сделано правильно, вы можете провести целую жизнь счастья с надежным и высокопроизводительным хостом, который всегда доступен по телефону, чату или электронной почте, чтобы ответить на ваши горящие ночные вопросы. Однако спешка в отношении выбора хостинга, без проведения исследования, может привести к тому, что вы окажетесь в ловушке, введены в заблуждение или даже стать жертвой вымогательства со стороны хостинга. Выбор неправильного хоста часто заканчивается головными болями и дорогим разводом.

Чтобы не переплачивать и быть обманутым надо чётко понимать, какой вид хостинга вам необходим. Если рассматривать, то все хостинги можно разделить на следующие виды.

Общий хостинг (виртуальный). В нём несколько клиентов и веб-сайтов используют один и тот же сервер. С одной стороны, виртуальный хостинг - это классический вид, с которого начинают развитие все малые и средние веб-приложения - простой и несложный. Не всегда, запуская веб-приложение малого или среднего бизнеса, можно выйти за рамки предоставляемые хостингом по памяти и ресурсам процессора. Лишь когда приложение набирает обороты необходимо решать, когда пора перейти на VPS или выделенный план для удовлетворения ваших растущих потребностей. Однако с другой точки зрения виртуальный хост ограничивает вас с сотнями или тысячами других. Поскольку ресурсы сервера распределены между очень многими сайтами, производительность иногда снижается по мере роста вашего сайта. Если вы готовы серьезно и реально увеличить свой трафик, вы, вероятно, не захотите соглашаться с общим планом хостинга.

Цена, поддержка, хранение и производительность - все это важные характеристики, которые следует учитывать при покупке общего хостинга. Другие отличия включают предложения электронной коммерции и бесплатные варианты

доменов, а также такие привилегии, как рекламные кредиты, конструктор сайтов и модернизированное оборудование.

VPS хостинг. Virtual private server (VPS) – является промежуточным звеном между общим хостингом и выделенным сервером. Сервер разделен на виртуальные машины, которые действуют как независимые выделенные серверы. Клиенты VPS по-прежнему имеют общий сервер, но у каждого из них гораздо большие ресурсов и больший контроль, чем у тех, у кого есть план общего хостинга.

Поскольку вы можете добавлять или удалять дополнительные вычислительные ресурсы по мере необходимости, планы хостинга VPS могут расширяться. У вас может быть довольно крупный основной сервер, но это не значит, что вы не можете подключить еще дополнительные мощностей в режиме ожидания.

Хосты VPS обычно включают в себя хранилище с высокоскоростными твердотельными накопителями или твердотельными накопителями, а также управляемые службы для обновления программного обеспечения. В зависимости от вашего уровня технического мастерства, вы можете пользоваться бесплатной панелью управления или, используя полный доступ прав root, настроить инфраструктуру самому. Вы также увидите, что топ-хосты VPS включают службы мониторинга, безопасности и CDN, чтобы держать вас осведомлённым.

Выделенный хостинг. Для высокопроизводительных сайтов требуется выделенный хостинг, что подразумевает использование всего сервера для питания вашего сайта или приложений. Как видно из названия, выделенные серверы готовы подождать с вами и идти навстречу и удовлетворить все ваши потребности в конфигурации. Клиенты имеют полный контроль над архитектурой, то есть они могут настраивать системы безопасности, операционные системы, балансировщики нагрузки и многое другое.

Однако это не обходится дешево. Выделенные планы хостинга являются одними из самых дорогих, учитывая первоклассное оборудование, управляемые услуги и круглосуточную поддержку. Однако хостинг высокого класса поставляется с рядом роскошных функций, включая автоматическую миграцию и резервное копирование, выделенные IP-адреса и выбор операционной системы.

2.2.5.2. Виды веб-приложений

Следующим этапом для выбора хостинга, надо знать какой тип веб-приложения вы планируете реализовать. Количество ожидаемого трафика или нагрузки

на сервер влияет на тип хостинг-плана, который вы хотите найти, а так же ваш тип сайта будет определять, какие функции наиболее важны. Например, некоторые хостинг-провайдеры продвигают функции электронной коммерции, в то время как другие концентрируются на блогах и поисковой оптимизации. Условно виды сайтов можно разделить на следующие группы.

Блоги. В связи с тем, что сайты на основе системы управления контента занимают более четверти рынка [**UsageSta21:online**; **NeedtoKn7:online**] среди всех веб-сайтов в Интернете, сайты блого-подобного характера стали простым выбором для авторов, желающих поделиться своими мыслями в Интернете. Сейчас каждый хост предлагает оптимизированные установки под них, но лучшие провайдеры включают в себя обновленное оборудование, неограниченное хранилище и пропускную способность, предустановленные программы, а также специальные знания и круглосуточную поддержку.

Интернет-магазины. В 2016 году аналитическая компания comScore обнаружила, что потребители покупают в Интернете больше вещей, чем в магазинах[**2016UPSP16:online**]. Более половины населения США совершают покупки в Интернете, поэтому компаниям следует найти веб-хостинга с широкими возможностями электронной коммерции. Ведущие хосты заботятся о дополнительных требованиях безопасности, связанных с защитой информации о клиентах и платежах, а также предоставляют красиво оформленные шаблоны, доступ к программному обеспечению корзины покупок и интеграцию с такими сервисами, как PayPal и инструменты почтового маркетинга.

Сайты портфолио (персональные сайты). Соискателям становится важно иметь сайт визитку в Интернете, даже если желаемая отрасль не имеет ничего общего с веб-дизайном или маркетингом. Связанно это с тем, что это самый быстрый способ создать профессиональное присутствие в Интернете, которое демонстрирует вашу работу (ваш бизнес). Отличаются такие сайты тем, что это отличный способ создания небольших сайтов с ограниченным бюджетом, в который можно относительно легко перевести в отрасль электронной коммерции.

Бизнес-сайты. Даже если вы не планируете использовать свой веб-сайт для продажи продуктов, ваш всё ещё бизнес рассчитывает на узнаваемость в Интернете как бренд. Предприниматели склонны ожидать, что их бизнес-сайт будет расти на 10-20% каждый месяц, если все идёт согласно их плану, грамотно выбранный хостинг-провайдер сможет справиться с быстро развивающимся бизнесом.

2.2.5.3. Виды ключевых ресурсов для веб-приложения

Определившись с типом веб-приложения, приходит понимание его особенностей и необходимых ресурсов. И тогда, выбирая хостинг, стоит ссылаться на соответствующие потребности веб-приложения, а не количестве предоставляемых услуг за меньшую плату. Например, компании могут отдавать предпочтение функциональности электронной почты над хранилищем, тогда как разработчик может предпочесть высокую пропускную способность и строгую безопасность. Если выделять технические особенности (ресурсы), то можно разбить их на следующие группы:

Доменные имена. Несмотря на то, что они обычно объединяются, регистрация доменов и веб-хостинг - это две разные услуги. Ваше доменное имя служит адресом вашего сайта и может быть зарегистрировано и размещено в компании, отличной от той, в которой размещены файлы вашего сайта. Держать все хостинговые ресурсы в одной хостинговой компании имеет в себе как и положительные так и негативные стороны. Из положительных можно выделить бесплатные переносы и миграции. Из негативных уменьшение уровня безопасности (но это правило работает не для всех хостингов).

Адреса электронной почты. Хостинг электронной почты особенно интересен владельцам бизнеса, которые хотят получить известность и признание имени, включив доменное имя в адреса электронной почты. Хостинг-провайдеры часто включают расширенные функции электронной почты, такие как службы пересылки и фильтрации, автоответчики и повышенную безопасность, для клиентов, которым требуется несколько почтовых ящиков или маркетинговых инструментов.

Память и оперативная память. Хранилище или дисковое пространство - это, пожалуй, самая простая для понимания функция хостинга, а также компонент, о котором вам, возможно, придется меньше всего беспокоиться. Многие провайдеры виртуального хостинга предлагают неограниченное хранилище; хотя это технически невозможно, большинство владельцев сайтов для частных лиц или предприятий малого бизнеса не приблизятся к достижению пределов. А когда вы переходите на VPS и выделенные планы, хранилище можно настраивать по мере необходимости.

Самая главное, на что нужно обращать внимание, когда дело доходит до хранилища веб-хостинга, это тип накопителя. Твердотельные накопители намного быстрее и надежнее, но стоят дороже. Традиционные жесткие диски, с другой

стороны, чаще встречаются, поскольку обычно они имеют более высокую емкость и привлекательны для хостингов.

Как и в случае с персональным компьютером, ОЗУ в веб-хостинге служит аналогичной цели: быстрая обработка хранимых данных. Аппаратное обеспечение взаимодействует с дисками хранения для ускорения загрузки страниц, и многие считают ОЗУ одной из самых важных функций, которые следует учитывать при выборе хоста.

Тарифы и надёжность. Каждая секунда недоступности сайта[**HowMuchD49:onl** может означать сотни упущенных возможностей продаж, подорвать репутацию бренда и потерю производительности. Отключение Amazon Web Services в начале 2017 года обошлось компаниям примерно в 150 миллионов долларов[**AmazonAn66:online**] за три-четыре часа прерванного обслуживания.

Некоторые хосты гарантируют определенное количество времени безотказной работы и возмещают вам любые незапланированные отключения, помимо соглашения об уровне обслуживания. Гарантии, как правило, варьируются от 100% до 99,9%. Хоть большинство клиентов виртуального хостинга будут удовлетворены порогом безотказной работы 99,9%, однако крупные компании предпочтут дополнительно заплатить за оставшиеся 0,1%.

Безопасность и поддержка. Безопасность веб-сайта в значительной степени зависит от производимой работы администратора, однако сама инфраструктура хостинговой компании может представлять одну из самых больших слабостей. Огромное количество веб-сайтов скомпрометированы из-за уязвимости хоста, поэтому обязательно, при поиске поставщиков услуг, стоит отдавать предпочтение тем, которые включают брандмауэры, службы мониторинга и другие надстройки безопасности. Большим плюсом будет ещё и автоматическое резервное копирование, если оно предоставляемо.

Клиентам хостинга будет сложно найти провайдера, который не предлагает круглосуточную поддержку, но реальное исполнение возникших проблем – сильно разнится. Независимо от того, простой ли вопрос или требуется техническая помощь, служба поддержки всегда должна быть готова помочь. К сожалению, этот критически важный аспект хостинга трудно разглядеть, пока вы не начнёте работать с ними и не обнаружите, что вам нужна помощь.

2.3. Выводы

Обладая чётким представлением какой тип веб-приложения разрабатываете и определив его ключевые ресурсы, можно уверено определить необходимый вид хостинга. С чётким представлением необходимого хостинга, остается только рассматривать просмотреть разные хостинг площадки на предмет необходимого плана или составления индивидуального плана.

Кроме всего выше перечисленного присутствует ещё один вариант размещения своего веб-приложения. Покупка собственного сервера. Такой вариант имеет как и очевидные плюсы, так и недостатки. Плюсы:

- Полный контроль. Создавая собственный сервер, вы контролируете все его аспекты: архитектуру (процессор, оперативная память, хранилище и его тип и так далее), ОС, библиотеки и так далее.
- Стоимость. Если ваше веб-приложение имеет долгосрочный характер, то цена на эксплуатацию складывается из стоимости компонентов и потребляемой электроэнергии. Кроме этого, на вашем сервере может существовать не одно приложение, что опять же сокращает расходы в перспективе (данное уточнение справедливо для веб-приложений малого и среднего бизнеса).

Недостатки:

- Полный контроль. Обладая полным контролем над всеми аспектами веб сервера и веб-приложения, не гарантирует наличие достаточной квалификации для грамотной настройки всей инфраструктурой в целом. Кроме того собственный сервер, для стабильной работы веб-приложения, обязует вас постоянно проверять как аппаратное, так и программное состояние на предмет уязвимостей. В свою очередь хостинг берёт на себя обязательства в проверке и поддержании аппаратной и частично программной части.
- Трафик. Для собственного сервера необходимо создавать дополнительный договор с провайдером. Обычного клиентского договора спокойно хватает на 20-50 одновременных пользователей, что достаточно для личного пользования. Однако для бизнеса необходимо b2b планы, так как размещение по существующему договору сайт будет слишком медленный для доступа.
- Безопасность. Собственный сервер может не иметь такой же уровень надёжности, как предлагаемый хостингами. При сбоях питания и широкополосного подключения ваш веб-сайт также будет отключен до тех

пор, пока не будет устранена неисправность. Так же самому необходимо обустраивать систему контроля резервных копий, которая предоставляется бесплатно в некоторых хостингах.

Размещение сайта на домашнем сервере может быть сделано, если вы знаете, что на самом деле делаете, а так же взвешены стоимость и выгода от этого. Если придётся тратить значительное время на изучение и покупку нового оборудования, обслуживание серверов и оплату увеличенных счетов за электроэнергию, возможно, нужно отказаться от этого, так как все ли эти проблемы.

Надо понимать, что недорогая служба веб-хостинга имеет все функции, которые могут быстрее доставлять веб-контент своим клиентам. Это избавляет от необходимости самостоятельно управлять прерываниями питания и решать другие проблемы.

ГЛАВА 3. РАЗРАБОТКА МЕТОДИКИ ПО СОЗДАНИЮ ВЫСОКОНАГРУЖЕННЫХ ВЕБ-ПРИЛОЖЕНИЯХ

3.1. Определение требований

В данной работе одной задачей является исследование основных источников угроз высоконагруженных веб-приложения. Основные угрозы для веб-приложений были рассмотрены и выявлены в первой главе настоящей работы. Так же обозначены специфичные угрозы для них.

Направлений разработки высоконагруженных веб-приложений – множество, каждое из них обладает своими потенциальными угрозами. Связанно это с тем, что высокие нагрузки – понятие относительное, как говорилось в первой главе. Однако все системы с высокой нагрузкой объединяет требовательность в ресурсах, как вычислительных, так и хранилищ. На основе этого сформируем методику разработки высоконагруженных систем, которая обобщит разные направления разработки, однако акцентирует внимание на обеспечении наименьшего показателя реакции системы.

3.2. Разработка рекомендаций

Основные угрозы, нарушающие корректную работоспособность веб-приложения, были описаны в первой главе текущей работы. Так же были определены основные виды данных приложений. Основываться на это и на результаты работы [**AmazonAn66:online**], а так же на проведённый анализ в рамках исследования, в целях минимизации уязвимостей рекомендуется:

- Продумать архитектуру, согласно которой будет реализовываться высоконагруженное веб-приложение;
- Создание маршрутной карты;
- Определится с механизмами аутентификации и авторизации;
- Проводить фильтрацию всей вводимой информации;
- Хранить данные в зашифрованном виде;
- Определить и использовать специализированные http заголовки;
- Ограничить SQL запросы;
- Реализовать взаимоисключающий доступ к ресурсам или директориям;
- Производить http опрос, для подтверждения наличия соединения.

3.3. Обоснование выбора инструментария

Для разработки прототипа приложения был определён следующий стек технологий.

Разработка серверной части:

Node js. В качестве серверного языка была выбрана платформа Node js. Основной причиной данного выбора является требованием со стороны заказчика. Однако для внесения ясности скажу, что node js основан на движке V8 от компании Google. Особенностью этого движка заключается в том, что он экономно расходует память, неплохо оптимизирован, дает функционал по профилированию процессора и памяти. Так же комьюнити по всему миру трудятся над ним, что в свою очередь стремительно улучает его.

Express.js – фреймворк, созданный для упрощения разработки для веб-приложений и API. Был выбран в качестве дополнения при разработке сервера. Express уже является стандартным каркасом для разработчиков веб-приложений, так как упрощает базовые функции предоставляемые из коробки Node js.

PostgreSQL была выбрана в качестве СУБД. Связанно это с тем, что она предоставляется по бесплатной лицензии в плоть до коммерческого использования, поддерживает сложные запросы (выходящие за рамки базовых SQL запросов), позволяет пользователям самим определять и писать свои функции и типы данных и это является одним из требования договора.

Socket.io. На основе исследования, проведенного во второй главе настоящей работы, для обеспечения взаимодействия между клиентами и сервером в режиме реального времени была выбрана библиотека socket.io. Она поддерживает передачу пользовательских событий, производит проверку на подключен ли пользователь, а так же производит автоматическое переподключение.

Для разработки клиентской части выбрали:

React – это библиотека с открытым исходным кодом для разработки пользовательского интерфейса. React в основном используется для разработки одностраничных и мобильных приложений. Связанно это с тем, что react реализует эффективную логику обновления узлов DOM. Происходит это из-за виртуального DOM (VDOM), где виртуальное представление пользовательского интерфейса хранится в памяти и синхронизируется с настоящим DOM при помощи процесса соглосования.

Easy-peasy. Библиотека, предоставляет вам интуитивно понятный API для быстрого и удобного управления состоянием вашего приложения. Обеспечивает хранение и управление состояний приложения в глобальном объекте (store).

Material-ui – это библиотека, которая позволяет создавать приложения в стиле Google Material Design с использованием компонентов React. Она упрощает веб-разработку, создание привлекательных пользовательских интерфейсов и одностраничных приложений.

3.4. Практическая реализация

Реализуем прототип приложения, который будет демонстрировать аспекты методики, чтобы произвести апробацию методики.

В качестве прототипа приложения для апробации методики было выбрано реализация специализированного приложения для формирования отчётов по данным с заездов парусных яхт в период тренировок. Цель приложения – получение табличных и графических отчётов.

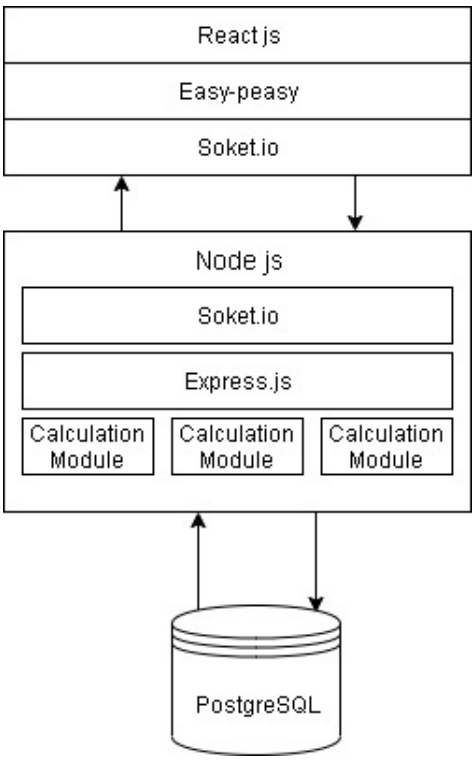

Рис.3.1. Архитектура приложения

Приложение должно обеспечивать возможность формировать отчёты на веб странице, а так же в формате pdf (Start report, XY report, Phases report, Performance report) по входящим данным. Кроме этого, хранить storage все рассчитанные данные для формирования отчётов на веб-странице.

Архитектура приложения Рис. 3.1 представляет собой взаимодействие серверной части на базе Node js, Express.js, базой данных PostgreSQL и стороннего модуля пред расчета данных и клиентской части, представленной React, библиотекой Material-ui, leaflet и хранилища на основе easy-peasy.

При подробном рассмотрении Рис. 3.1, как ранее было сказано, библиотека React представляет клиентскую часть. Она обеспечивает масштабируемую разработку пользовательского интерфейса. Централизованное хранение информации реализовано с помощью библиотеки Easy-peasy. Она представляет информацию в виде глобального объект-хранилища. Взаимодействие с серверной частью происходит в рамках классического RESTful API. Реализовано с помощью отправки со стороны клиента Ajax-запросов, которые реализованы в Socket.io. Так же сам трансфер данных реализован с помощью Socket.io в рамках шаблона проектирования «издатель-подписчик». Это позволяет клиенту передавать данные заезда парусной яхты для расчёта отчётов на серверной стороне, которые он сохраняет в базе данных и отправляет обработанные данные клиенту для формирования отчётов на клиентской стороне. Для взаимодействия с PostgreSQL используется

модуль pg[**pgnpm78:online**] (неблокирующий клиент PostgreSQL для Node.js, не требующий дополнительных библиотек), позволяющий производить манипуляцию данных между базой данных и сервером. При получении данных на сервер определенного типа о необходимости расчёта данных, первоначально осуществляется поиск уже обработанных данных в СУБД PostgreSQL. Если же данные не найдены, то осуществляется расчёт отправленных пользователем данных.

На Рис. 3.2 представлен пользовательский интерфейс приложения, разработанный с помощью Material-UI, а так же отчёт по Start report, показывающий инфографику начала стартового пути яхты.

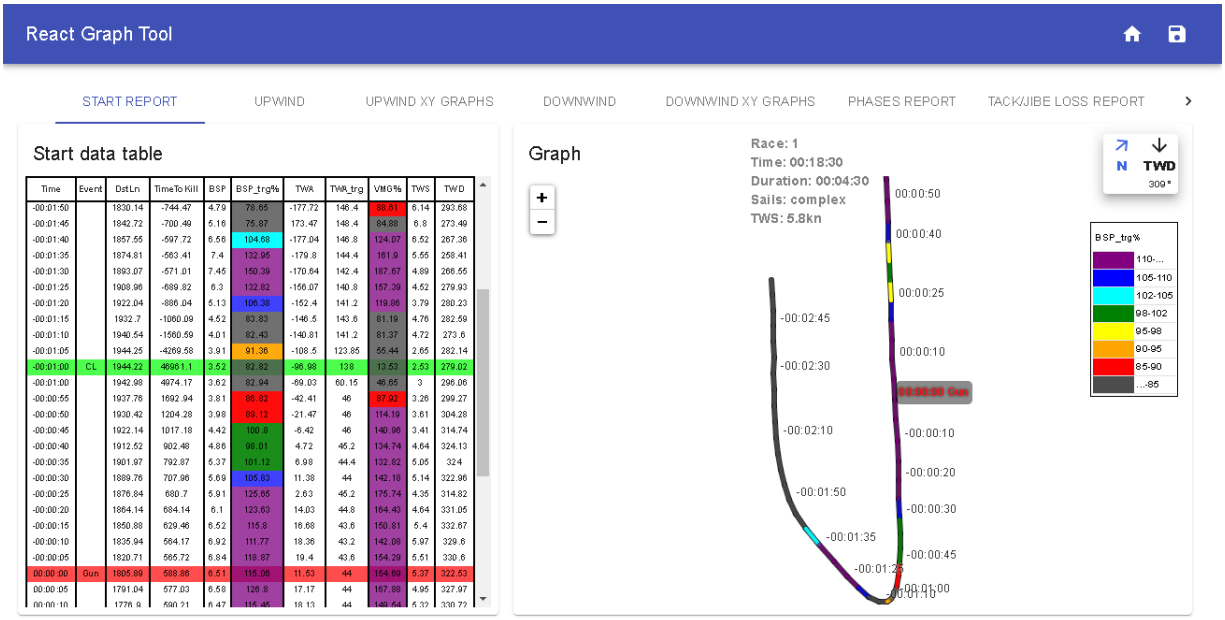

Рис.3.2. Пользовательский интерфейс вкладки Start report

На Рис. 3.3 представлен, разработанный Upwind xy report, инфографику показывающие отношение характеристик к угловому ветру с помощью Material-UI.

На Нис.3.4 представлен срез глобального хранилища Easy-peasy.

Приложения [**appendix:RGTMapGraph**; **appendix:RGTTableGraph**; **appendix:ReliseRESTfulAPI**] демонстрируют код моделей табличного представления данных, графа пути.

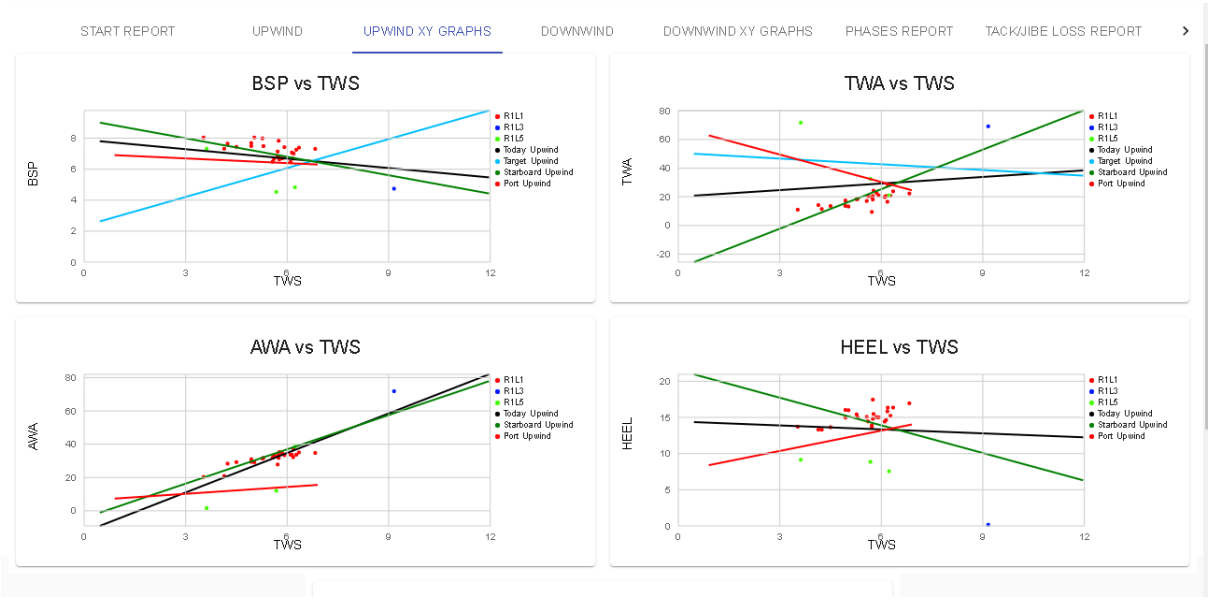

Рис.3.3. Upwind XY report

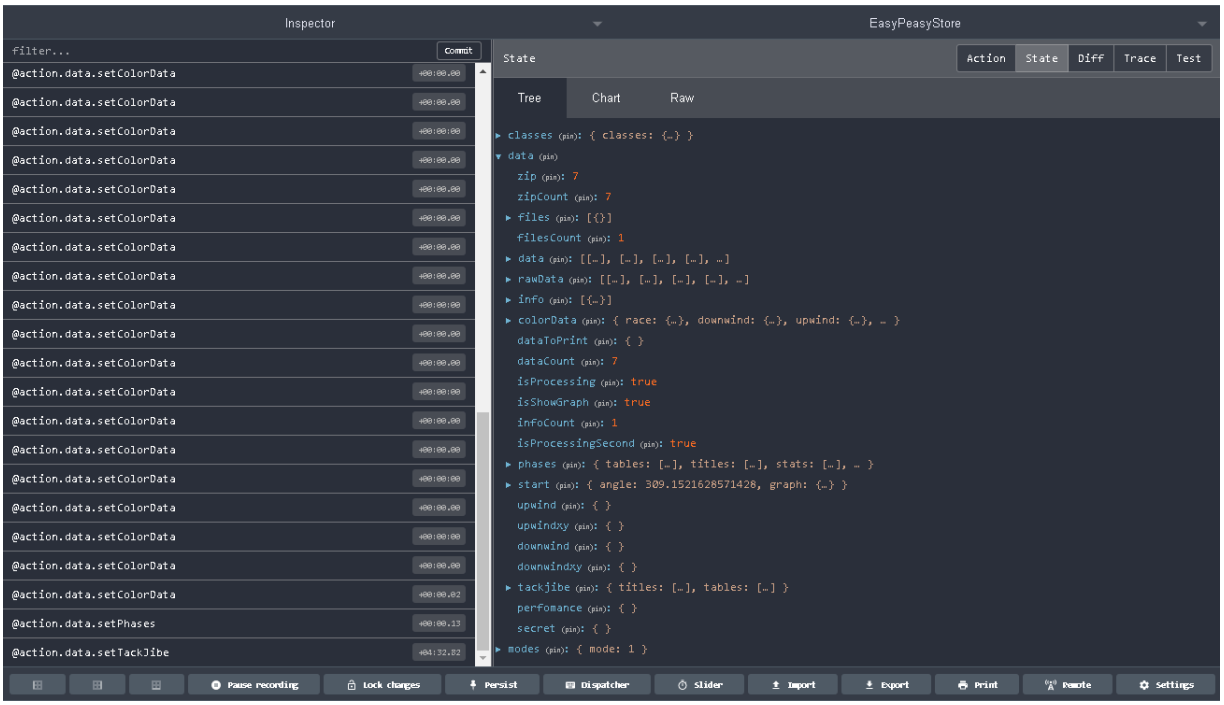

Рис.3.4. Upwind XY report

3.5. Сравнение с существующими технологиями

Тестирование прототипа реализовано с помощью фреймворка для нагрузочного тестирования Artillery.io. Он позволяет писать сценарии взаимодействия с приложением, а так же имитировать сложное поведение пользователя посредством нескольких шагов и транзакций с помощью циклов, условий на коде Javascript. Artillery.io удобно использовать для отладки сложных приложений, таких как транзакционные API, бэкэнд IoT,, бэкэнды онлайн игр и все виды сервисов с отслеживанием состояния.

Artillery.io обладает широким функционалом, который позволяет оценить время отклика, время задержки, количество запросов в секунду или отслеживать кастомные показатели с помощью кода Javascript.

Пример одного из использованных пользовательских сценариев показан:

```
{
config:
    target: "https://localhost:3000"
    phases:
  - duration: 1000
      arrivalRate: 1
    payload:
      path: "Report_2020_05_13_11_59_01.fsr"
      fields:
      - "files"
scenarios:
  - name: "Show start graph"
    flow:
    - post:
        url: "/api/start-report"
        body: "file={{ keywords }}"
    - get:
        url: "/start-report"
    - think: 500
```

Каждую секунду в течение 1000 секунд новый виртуальный пользователь отправляет файл из поля fields.file, пост запросом, на отображение стартовой страницы и остаётся в открытом соединении в течение 500 секунд.

Artillery.io предоставляет множество метрик для оценки тестов. Одни из них - количество выполненных запросов. Они являются количеством отправленных HTTP-запросов и ответов или сообщений от WebSocket. Так же число отправленных запросов в секунду. Является средним количеством запросов в секунду,

Таблица 3.1

Результаты нагрузочного тестирования

	Request latency (min)	Request latency (max)	Request latency (median)	P95	P99
B&G sailing features	121 мс	135 мс	127 мс	123 мс	133 мс
Sailwave	(только Desktop)	(только Desktop)	(только Desktop)	(только Desktop)	(только Desktop)
Прототип приложе-ния	128 мс	169 мс	143 мс	132 мс	144 мс

выполненных за предыдущие 10 секунд (или в течение всего теста). Задержка запроса в миллисекундах. Значения p95 и p99, которые показывают максимальный предел задержки у 99 и 95 соответственно процентов запросов (к примеру, для 95 - показывает, что из 100 запросов 95-и запросам потребовалось меньше или равное 450мс).

Для сравнения с аналогами было принято решение использовать фреймворк artillery.io для проведения нагрузочного тестирования приложений B&G sailing features и Sailwave на основе API Pusher.

Таблица 3.2

Таблица сравнения программ

	Вид распространения	Цена	Функционал
B&G sailing features	Desktop/ lite web	От 700$ и может больше 2000$ в зависимости от решения	Start report, Phase report, XY report, Tack/Jibe Loss report
Sailwave	Desktop	Free	Phase report, XY report, Tack/Jibe Loss report
Прототип приложения	web	Free	Start report, Phase report, XY report, Tack/Jibe Loss report, Perfomances

3.6. Выводы

В данной главе была сформулирована методика, предполагающая выполнение последовательности действий для сокращения угроз. Так же был определен стек для разработки прототипа приложения. Прототип специализированного веб-приложения для формирования отчётов по данным с заездов парусных яхт в период тренировок был разработан. Так же данный прототип подвергся нагрузочному тестированию. Были созданы тесты, которые имитируют посещение веб-приложения виртуальными пользователями. Нагрузочное тестирование показало, что прототип уступает по показателям, однако результаты можно считать сопоставимыми с результатами коммерческие аналогов, из чего следует, что предложенный архитектурный подход можно считать приемлемым.

ЗАКЛЮЧЕНИЕ

В первой главе была произведена классификация highload приложений и они были определены. Были представлены направления, в которых используется высоконагруженные системы и примеры таких систем. Были проанализированы актуальные угрозы веб-приложений, а так же методы борьбы с ними.

Во второй главе были определены тип веб-приложения, ключевые ресурсы веб-приложений и виды хостинга. Кроме того, были проанализированы виды транспорта передачи данных с описанием достоинства и недостатков каждого. Были определены инструменты для построения таких веб-приложений. Было проведено исследование задержки реакции системы при разработке используя стандартный протокол WebSocket, Socket.io и SockJS. Проведён эксперимент по замеру времени передачи сообщения.

В данной главе была сформулирована методика, предполагающая выполнение последовательности действий для сокращения угроз. Так же был определен стек для разработки прототипа приложения. Прототип специализированного веб-приложения для формирования отчётов по данным с заездов парусных яхт в период тренировок был разработан. Так же данный прототип подвергся нагрузочному тестированию. Были созданы тесты, которые имитируют посещение веб-приложения виртуальными пользователями. Нагрузочное тестирование показало, что прототип уступает по показателям, однако результаты можно считать сопоставимыми с результатами коммерческие аналогов, из чего следует, что предложенный архитектурный подход можно считать приемлемым.

На основании данных результатов, можно утверждать, что прототип веб–приложения, реализованный в рамках методики может составить конкуренцию аналогичным коммерческим продуктам, так как является более предпочтительным по некоторым аспектам, например, из-за цены такого подхода.

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

DOI Digital Object Identifier.

WoS Web of Science.

ВКР Выпускная квалификационная работа.

ТГ-объект Текстово-графический объект.

APM Application Performance Monitoring

СЛОВАРЬ ТЕРМИНОВ

TeX — язык вёрстки текста и издательская система, разработанные Дональдом Кнутом.

LaTeX — язык вёрстки текста и издательская система, разработанные Лэсли Лампортом как надстройка над TeX.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Приложение 1

Реализация метода анализа иерархий (Саати)

Листинг кода метода Саати для выбора наилучшего решения из представленных вариантов по заданным критериям.

main.py:

```
import json
import my_math as math

model_tmp_table = {}
text = ''

with open('model2.json') as json_model:
# model can also be a python dictionary
model = json.load(json_model)

text = math.makeSignificanceTable(model_tmp_table, model, text
    )
text = math.makeOptionMatrixTables(model_tmp_table, model,
    text)
text = math.makeFinaleMatrix(model_tmp_table, model, text)

answer = model_tmp_table['result']

list_d = list(answer.items())
list_d.sort(key=lambda i: i[1])
last_item = len(list_d)-1

for i in list_d:
print('{0:<10} : {1:^7f}'.format(i[0],i[1]))
text += '{0:<10} : {1:^7f}'.format(i[0],i[1]) + "\n"

print('\nBest item is => {:<20}'.format(list_d[last_item][0]))
text += '\nBest item is => {:<20}'.format(list_d[last_item
    ][0])

f= open("result.txt","w+")

f.write(text)
f.close()
```

my_math.py:

```
import numpy as np
import json
import copy

def format_matrix(header, matrix, top_format, left_format,
    cell_format, row_delim, col_delim, val, kek = False, pos =
    0):

if(kek):
header.insert(pos, "")
for i in matrix:
i.insert(pos, "")

table = [['']  + header] + [[name] + row for name, row in zip(
    header, matrix)]
if(kek):
table_format = [['{:^{}}'] + len(header) * [top_format]] + (
    len(matrix)  - 2 + val) * [[left_format] + (len(header) -
    3) * [cell_format] + ['{}'] + 2 * [cell_format]]
else:
table_format = [['{:^{}}'] + len(header) * [top_format]] + (
    len(matrix)  - 2 + val) * [[left_format] + len(header) * [
    cell_format]]
col_widths = [max(len(format.format(cell, 0)) for format, cell
     in zip(col_format, col))for col_format, col in zip(zip(*
    table_format), zip(*table))]
return row_delim.join(col_delim.join(format.format(cell, width
    )
for format, cell, width in zip(row_format, row, col_widths))
for row_format, row in zip(table_format, table))



def makeSignificanceTable(model_tmp_table, model, my_str): #
    Second level matrix
significance_table = {}
criteria = model['criteria']
for idx1, key1 in enumerate(criteria):
row = []
for idx2, key2 in enumerate(criteria):
row.append(criteria[key1]/criteria[key2])
significance_table[key1] = row

for row in significance_table:
tmp = 1
```

```
    for j in significance_table[row]:
70  tmp *= j
    tmp = tmp**(1./len(significance_table[row]))
    significance_table[row].append(tmp)

    tmp = 0
75  for row in significance_table:
    tmp += significance_table[row][len(significance_table[row]) -
       1]
    check = 0

    length = len(list(significance_table.values())[0]) - 1
80  for row in significance_table:
    significance_table[row].append(significance_table[row][length]
        / tmp)
    check += significance_table[row][length + 1]

    str = 'Significance Table'
85  print("\n"+"#"*(6 + len(str))+"\n\n   " + str + "\n\n"+"#"*(6
       + len(str)))
    my_str += "\n"+"#"*(6 + len(str))+"\n\n   " + str + "\n\n
       "+"#"*(6 + len(str)) + "\n"
    header = [key for key in significance_table]
    matrix = [significance_table[key] for key in
       significance_table]
    print(format_matrix(header, matrix, '{:^{}}', '{:<{}}',
       '{:>{}.5f}', '\n', ' | ', 2))
90  my_str += format_matrix(header, matrix, '{:^{}}', '{:<{}}',
       '{:>{}.5f}', '\n', ' | ', 2) + "\n\n\n"

    print("\n\n")

    model_tmp_table['significance_criteria'] = significance_table
95  return my_str



    def makeOptionMatrixTables(model_tmp_table, model, my_str): #
       Third level matrixs
    matrix = {}
100 matrix_by_options = model['options']
    criteria_names = model['criteria'].keys()
    tmp_list = matrix_by_options.values()
    length = len(list(tmp_list)[0])

105 for i in range(length):
```

```
table = []
for idx1, key1 in enumerate(matrix_by_options):
row = []
for idx2, key2 in enumerate(matrix_by_options):
row.append(matrix_by_options[key1][i] / matrix_by_options[key2
   ][i])
table.append(row)

matrix[list(criteria_names)[i]] = table

for idx, key in enumerate(matrix):
table = matrix[key]
for row in table:
tmp = 1
for j in row:
tmp *= j
tmp = tmp**(1./len(row))
row.append(tmp)

for idx, key in enumerate(matrix):
tmp = 0
for row in matrix[key]:
tmp += row[len(row) - 1]
check = 0
length = len(matrix[key][0]) - 1
for row in matrix[key]:
row.append(row[length] / tmp)
check += row[length + 1]

str1 = 'Tables of significant factors for each criterion'
print("\n"+"#"*(6 + len(str1))+"\n\n   " + str1 + "\n\n
   "+"#"*(6 + len(str1)))
my_str += "\n"+"#"*(6 + len(str1))+"\n\n   " + str1 + "\n\n
   "+"#"*(6 + len(str1)) + "\n"
header = [key for key in matrix_by_options]

for key in matrix:
print("\n\tBy "+ key)
my_str += "\n\tBy "+ key + "\n"
tmp = 0
for row in matrix[key]:
tmp += row[len(row) - 1]
check = 0
length = len(matrix[key][0]) - 1
for row in matrix[key]:
```

```
row.append(row[length] / tmp)
check += row[length + 1]
answ = format_matrix(copy.deepcopy(header) + ["Weight", "% of
    importance"], copy.deepcopy(matrix[key]) , '{:^{}}',
    '{:<{}}', '{:>{}.5f}', '\n', ' | ', 2, True, len(header))
print(answ)
my_str += answ + "\n"
print("SUM %: ", check)
my_str += "SUM %: " + str(check) + "\n"
print("\n\n")
model_tmp_table['matrix_by_criteria'] = matrix
return my_str



def makeFinaleMatrix(model_tmp_table, model, my_str): # Final
    result
final = []
for_multiply = model_tmp_table["significance_criteria"]
conclusion = model_tmp_table["matrix_by_criteria"]

str = 'Final matrix for calculating weights for each element'

fin_answ_tab = "\n"+"#"*(6 + len(str))+"\n\n   " + str + "\n\n
    "+"#"*(6 + len(str)) + "\n\n"

header = [key for idx, key in enumerate(conclusion)]
str_leng = max([len(key) for idx, key in enumerate(conclusion)
    ])
ttt = max([len(key) for idx, key in enumerate(model['options
    '])])
str_leng = max(str_leng, ttt)
str_leng =  8 if str_leng<8 else str_leng
side = [key for idx, key in enumerate(model['options'])]
side.insert(0, " " * str_leng)

for i, row in enumerate(side):
if i == 0:
for idx, key in enumerate(conclusion):
row += "|"
row += "{name:<{size}}".format(name=key, size=str_leng)
row += "|" +  "{name:<{size}}".format(name="% weight", size=
    str_leng) + "\n"
row += "-" * str_leng + "|" + "|".join([("-" * str_leng) for i
     in range(len(conclusion) + 1)]) + "\n"
else:
```

```
row = "{name:<{size}}".format(name=row, size=str_leng)
summ_to_final = 0
for idx, key in enumerate(conclusion):
summ_to_final += copy.deepcopy(conclusion[key][i-1]).pop() *
   copy.deepcopy(for_multiply[key]).pop()
row += "|" + "{name:>{size}.5f}".format(name=copy.deepcopy(
   conclusion[key][i-1]).pop(), size=str_leng)

row += "|" + "{name:>{size}.5f}".format(name=summ_to_final,
   size=str_leng) +"\n"
fin_answ_tab += row
fin_answ_tab += "-" * str_leng + "|" + "|".join([("-" *
   str_leng) for i in range(len(conclusion))]) + "|" + "\n"
fin_answ_tab += "{name:^{size}}".format(name="SP", size=
   str_leng) + "|"
for i, key in enumerate(for_multiply):
fin_answ_tab += "{name:>{size}.5f}".format(name=copy.deepcopy(
   for_multiply[key]).pop(), size=str_leng) + "|"

fin_answ_tab += "\n\n\t Where SP - the significance of the
   criterion\n\n"
print(fin_answ_tab)
for idx, key in enumerate(conclusion):
row = []
length = len(for_multiply[key]) - 1
for i in conclusion[key]:
row.append(i[len(i)-1] * for_multiply[key][length])
final.append(row)

arr1 = np.array(final)
arr1_transpose = arr1.transpose()
okey = {}
x = iter(list(model["options"].keys()))
for i in arr1_transpose:
k = next(x)
okey[k] = sum(i)

model_tmp_table['result'] = okey
return my_str + fin_answ_tab
```

Приложение 2

Расчёт наилучшего транспорта передачи данных

Зададим значимость к критерия для расчётов. Количество запросов на опрос сервера (Cnt req on poll): 4. Количество служебных заголовков вместе с ответом (Cnt of serv head): 3. Служебная информация прикрепляемая в ответе и запросе (Service info): 5. Согласно табл.2.1, модель соответствия вариантов критериям имеет следующие значения:

```
{
  "name": "Choose transfer data",
  "criteria": {
    "Cnt req on poll"   : 4,
    "Cnt of serv head"  : 3,
    "Service info"      : 5
  },
  "options": {
    "WebSockets"          :   [4, 4, 5],
    "Server-sent events" :   [4, 4, 4],
    "Long Polling"        :   [4, 3, 2],
    "Ajax Polling"        :   [3, 3, 2]
  }
}
```

Расчёт наилучшего решения приведённой модели из Приложения 1:

```
##################################

   Significance Table

##################################
                  | Cnt req on poll | Cnt of serv head | Service info
Cnt req on poll   |         1.00000 |          1.33333 |      0.80000
Cnt of serv head  |         0.75000 |          1.00000 |      0.60000
Service info      |         1.25000 |          1.66667 |      1.00000
```

Рис.П2.1. Таблица значимости

```
##################################################################

    Tables of significant factors for each criterion

##################################################################

          By Cnt req on poll
          | WebSockets | Server-sent events | Long Polling | Ajax Polling |  | Weight  | % of importance |
WebSockets |    1.00000 |            1.00000 |      1.00000 |      1.33333 |  | 1.07457 |         0.26667 |
Server-sent events |    1.00000 |            1.00000 |      1.00000 |      1.33333 |  | 1.07457 |         0.26667 |
Long Polling |    1.00000 |            1.00000 |      1.00000 |      1.33333 |  | 1.07457 |         0.26667 |
Ajax Polling |    0.75000 |            0.75000 |      0.75000 |      1.00000 |  | 0.80593 |         0.20000 |
SUM %: 1.0

          By Cnt of serv head
          | WebSockets | Server-sent events | Long Polling | Ajax Polling |  | Weight  | % of importance |
WebSockets |    1.00000 |            1.00000 |      1.33333 |      1.33333 |  | 1.15470 |         0.28571 |
Server-sent events |    1.00000 |            1.00000 |      1.33333 |      1.33333 |  | 1.15470 |         0.28571 |
Long Polling |    0.75000 |            0.75000 |      1.00000 |      1.00000 |  | 0.86603 |         0.21429 |
Ajax Polling |    0.75000 |            0.75000 |      1.00000 |      1.00000 |  | 0.86603 |         0.21429 |
SUM %: 1.0

          By Service info
          | WebSockets | Server-sent events | Long Polling | Ajax Polling |  | Weight  | % of importance |
WebSockets |    1.00000 |            1.25000 |      2.50000 |      2.50000 |  | 1.67185 |         0.38462 |
Server-sent events |    0.80000 |            1.00000 |      2.00000 |      2.00000 |  | 1.33748 |         0.30769 |
Long Polling |    0.40000 |            0.50000 |      1.00000 |      1.00000 |  | 0.66874 |         0.15385 |
Ajax Polling |    0.40000 |            0.50000 |      1.00000 |      1.00000 |  | 0.66874 |         0.15385 |
SUM %: 1.0
```

Рис.П2.2. Таблицы значимых факторов для каждого критерия

```
######################################################################

   Final matrix for calculating weights for each element

######################################################################
```

	Cnt req on poll	Cnt of serv head	Service info	% weight
WebSockets	0.26667	0.28571	0.38462	0.32057
Server-sent events	0.26667	0.28571	0.30769	0.28852
Long Polling	0.26667	0.21429	0.15385	0.20656
Ajax Polling	0.20000	0.21429	0.15385	0.18434
SP	0.33333	0.25000	0.41667	

```
         Where SP - the significance of the criterion

Ajax Polling : 0.184341
Long Polling : 0.206563
Server-sent events : 0.288523
WebSockets : 0.320574

Best item is => WebSockets
```

Рис.П2.3. Результирующя таблица

Приложение 3

Сравнение выполнения одинаковых задач разными языками программирования

Данная таблица построена на основе данных с электронного ресурса [**\ref {appendix:methodSaati}**] . Цветовая идентификация имеет следующий характер: Насыщенный зелёный – самый лучший результат, зелёный – второй лучший результат, оранжевый – самый худший результат.

Таблица П3.1

Список угроз с основными методами противодействия им

	PHP	Node js	Java	Python
regex-redux	2,88	9,53	10,27	2,67
pidigits	2,11	2,58	1,83	2,39
k-nucleotide	41,32	26,32	9,14	72,58
binary-trees	51,13	19,22	8,28	80,82
reverse-complement	14,19	4,1	3,27	16,41
spectral-norm	33,24	4,4	4,15	170,1
fannkuch-redux	219,15	19,48	16,12	494,58
n-body	329,52	26,28	21,85	891,12
fasta	53,24	3,62	2,22	63,63
mandelbrot	105,4	6,84	6,84	263,87

Приложение 4

Выбор серверного языка

Зададим значимость к критерия для расчётов. Согласно Таблица 2.2., модель соответствия вариантов критериям имеет следующие значения:

```
{
  "name": "Choose best language",
  "criteria": {
    "Libraries" : 4,
    "DB"        : 3,
    "Speed"     : 5,
    "Framework" : 4
  },
  "options": {
    "Python"  :   [5, 3, 3, 4],
    "Node Js" :   [5, 4, 4, 5],
    "Java"    :   [4, 5, 5, 4],
    "PHP"     :   [4, 4, 4, 4]
  }
}
```

Расчёт наилучшего решения приведённой модели с Приложение 1

```
############################################################

   Final matrix for calculating weights for each element

############################################################

         |Libraries|DB       |Speed    |Framework|% weight
---------|---------|---------|---------|---------|---------
Python   |  0.27778|  0.18750|  0.18750|  0.23529|  0.22202
Node Js  |  0.27778|  0.25000|  0.25000|  0.29412|  0.26797
Java     |  0.22222|  0.31250|  0.31250|  0.23529|  0.27063
PHP      |  0.22222|  0.25000|  0.25000|  0.23529|  0.23938
---------|---------|---------|---------|---------|
   SP    |  0.25000|  0.18750|  0.31250|  0.25000|

         Where SP - the significance of the criterion


Python     : 0.222018
PHP        : 0.239379
Node Js    : 0.267974
Java       : 0.270629

Best item is => Java
```

Рис.П4.1. Результирующая таблица

Приложение 5

Реализация серверной части Socket.io

Листинг программного кода серверной части на Node.js, Express.js и Socket.io, для анализа библиотек.

```
const path = require('path');
const express = require('express');

const port = process.env.PORT || 80;
const app = express();
const server = require('http').Server(app);
const socketio = require('socketio')(server);

const uuid = require('uuid/v4');
const createCsvWriter = require('csv-writer').
   createObjectCsvWriter; const csvWriter = createCsvWriter({
```

```
path: 'stats-si.csv',
header: [
{id: 'msg', title: 'MSG'},
{id: 'id', title: 'UUID'},
{id: 'tmspStart', title: 'TMPSTART'},
{id: 'tmspEnd', title: 'TMPEND'}
]
});
const messageCount = 1000;
const stats = new Map();

io.on('connection', socket => {
for (let t = 0; t < messageCount; t++) {
const id = uuid();
const sendedMsg = { msg: 'ping', id, tmspStart: Date.now(),
   tmspEnd: ''};
stats.set(sendedMsg.id, sendedMsg); io.sockets.emit('message',
    JSON.stringify(sendedMsg));
}

socket.on('message', (message) => {
if (counter < messageCount) {
counter++;
} else {
io.close();
}
const parsedMsg = JSON.parse(message);
parsedMsg.tmspEnd = Date.now();
parsedMsg.msg = "roundtrip";
parsedMsg.tmspStart; stats.set(parsedMsg.id, parsedMsg);
});

socket.on('disconnect', () => {
const records = Array.from(stats, keyValuePair => keyValuePair
   [1]);
csvWriter.writeRecords(records)
.then(() => {
})
.catch((err) => { console.log(err) });
})
});

app.get('/', function (req, res) {
res.sendFile(path.resolve('../index.html'));
});
```

```
55 server.listen(port, () => console.log(`Listening on port ${
     port}`));
```

Приложение 6

Реализация серверной части SockJS

Листинг программного кода серверной части на Node.js, Express.js и Socket.io, для анализа библиотек.

```
const http = require('http');
const sockjs = require('sockjs');
const node_static = require('node-static');
const path = require('path'); const fs = require("fs");
const uuid = require('uuid/v4');
const createCsvWriter = require('csv-writer').
   createObjectCsvWriter; const csvWriter = createCsvWriter({
path: 'stats-sk.csv',
header: [
{id: 'msg', title: 'MSG'},
{id: 'id', title: 'UUID'},
{id: 'tmspStart', title: 'TMPSTART'},
{id: 'tmspEnd', title: 'TMPEND'}
]
});

const sockjs_opts = {
prefix: '/echo'
};

const port = 80;
const messageCount = 1000;
let counter = 0;
const stats = new Map();
const sockjs_echo = sockjs.createServer(sockjs_opts);
   sockjs_echo.on('connection', function(conn) {
for (let t = 0; t < messageCount; t++) {
const id = uuid();
const sendedMsg = { msg: 'ping', id, tmspStart: Date.now(),
   tmspEnd: ''};
stats.set(sendedMsg.id, sendedMsg);
conn.write(JSON.stringify(sendedMsg));
}
conn.on('data', (message) => {
if (counter < messageCount) {
counter++;
} else {
```

```
conn.close();
}
const parsedMsg = JSON.parse(message);
parsedMsg.tmspEnd = Date.now();
parsedMsg.msg = "roundtrip";
stats.set(parsedMsg.id, parsedMsg);
});
conn.on('close', function() {
console.log('connection is closed');
const records = Array.from(stats, keyValuePair => keyValuePair
   [1]);
csvWriter.writeRecords(records);
});
});

const static_directory = new node_static.Server(path.resolve
   ('../test-ws/client/index-sk.html'));

const server = http.createServer(); server.addListener('
   request', function(req, res) {
static_directory.serve(req, res);
});

server.addListener('upgrade', function(req, res) {
res.end();
});
sockjs_echo.installHandlers(server);
console.log(` Server listening on ${port}`);
server.listen(port, hostname);
```

Приложение 7

Выбор систему мониторинга

Зададим значимость к критерия для расчётов. Согласно Таблица 2.3., модель соответствия вариантов критериям имеет следующие значения:

```
{
  "name": "Choose best APM",
  "criteria": {
    "UI"                                : 3,
    "Distribution"                      : 4,
    "Tracing transaction"               : 5,
    "Correlation data"                  : 3,
     "Support"                           : 4
  },
  "options": {
    "New Relic"      :    [5, 3, 4, 5, 5],
    "CloudWatch"     :    [3, 4, 1, 3, 3],
    "Dynatrace APM"  :    [4, 5, 5, 5, 4],
    "AppDynamics"    :    [5, 5, 4, 4, 5],
    "CA APM"         :    [4, 5, 4, 5, 4]
 }
}
```

Расчёт наилучшего решения приведённой модели с Приложение 1:

```
############################################################

   Final matrix for calculating weights for each element

############################################################

                  |UI               |Distribution       |Tracing transaction|Correlation data   |Support            |% weight
------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------
New Relic         |           0.23810|           0.13636|           0.22222|           0.22727|           0.23810|           0.21079
CloudWatch        |           0.14286|           0.18182|           0.05556|           0.13636|           0.14286|           0.12706
Dynatrace APM     |           0.19048|           0.22727|           0.27778|           0.22727|           0.19048|           0.22701
AppDynamics       |           0.23810|           0.22727|           0.22222|           0.18182|           0.23810|           0.22275
CA APM            |           0.19048|           0.22727|           0.22222|           0.22727|           0.19048|           0.21239
------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
       SP         |           0.15789|           0.21053|           0.26316|           0.15789|           0.21053|

        Where SP - the significance of the criterion


CloudWatch : 0.127060
New Relic  : 0.210792
CA APM     : 0.212387
AppDynamics : 0.222754
Dynatrace APM : 0.227007

Best item is => Dynatrace APM
```

Рис.П7.1. Результирующая таблица

Приложение 8

Компонента Map (граф)

Листинг кода реализации компоненты Map (граф начального пути яхты в start report)

```
const MyMap = ({ graph, title, angle }) => {
const { classes } = useStoreState(state => state.classes);
const { rawData } = useStoreState(state => state.data);
const [lastZoom, changeLastZoom] = useState(0);
let heap = 0;
const mapRef = useRef(null);

const hideAndSeek = event => {
  let map = event.target;
  heap = 0;
  var zoom = map.getZoom();
  if (zoom < 16 && (!lastZoom || lastZoom >= 16)) {
    map.eachLayer(function(l) {
      if (l.getTooltip()) {
        var tooltip = l.getTooltip();
        l.unbindTooltip().bindTooltip(tooltip, {
          permanent: false
        });
      }
    });
  } else
  map.eachLayer(function(l) {
    if (l.getTooltip()) {
      var tooltip = l.getTooltip();
      l.unbindTooltip().bindTooltip(tooltip, {
        permanent: labelPermanent(tooltip.options.elevation,
           zoom)
      });
    }
  });

  changeLastZoom(zoom);
};

const labelPermanent = (wheight, zoom) => {
  let denominator = zoom - 16;
  let border = 320 / Math.pow(2, denominator);
```

```
  if (heap + wheight > border) {
    heap = 0;
    return true;
  } else {
    heap = heap + wheight;
    return false;
  }
};

const log = num => {
  return Math.log(num) / Math.log(2.0154);
};

const takeZoomLvl = () => {
  let zoomLvl = {
    x: [Math.min(...graph.x), Math.max(...graph.x)],
    y: [Math.min(...graph.y), Math.max(...graph.y)]
  };
  let y = Math.abs(zoomLvl.y[1] - zoomLvl.y[0]);
  let x = Math.abs(zoomLvl.x[1] - zoomLvl.x[0]);
  return Math.min(log(720 / x), log(360 / y));
};
return (
  <>
  {graph.hasOwnProperty("myGeoJSON") &&
  graph.hasOwnProperty("myGeoJSONLable") ? (
    <>
      <Legend />
      <Graphtitle tws={graph.tws} />
      <Arrow angle={angle} />
      {title && <Title title={title} />}
      <Map
        className={classes.mapLeaflet}
        id="leaflet-map"
        center={[graph.center[1], graph.center[0]]}
        useFlyTo={true}
        boundsOptions={{ padding: [25, 25] }}
        zoom={takeZoomLvl()}
        attributionControl={true}
        zoomControl={true}
        doubleClickZoom={true}
        scrollWheelZoom={true}
        dragging={true}
        maxZoom={20}
        minZoom={1}
```

```
        animate={true}
        easeLinearity={0.35}
        onZoomend={e => hideAndSeek(e)}
        ref={mapRef}
      >
        <TileLayer url="http://{s}.tile.osm.org/{z}/{x}/{y}.
           png" />
        {graph.geoJsonLines.features.map((feature, idx) => (
          <GeoJSON
            key={feature.key}
            data={feature}
            style={getStyle(feature.properties.name)}
          >
            {feature.properties.hasOwnProperty("point") ? (
              feature.properties.point.name.map((name, jdx) =>
                  (
                <Marker
                  position={feature.properties.point.
                     coordinates[jdx]}
                  key={"SL-" + feature.properties.point.name[
                     jdx]}
                >
                  <Tooltip
                    key={
                      "Tooltip-" + feature.properties.point.
                         name[jdx] + jdx
                    }
                    direction="right"
                    offset={[-25, 15]}
                    elevation={feature.properties.point.
                       elevation}
                    className={"tooltopSP"}
                    permanent={labelPermanent(
                    feature.properties.point.elevation,
                    takeZoomLvl()
                    )}
                  >
                    {feature.properties.point.name[jdx]}
                  </Tooltip>
                </Marker>
              ))
              ) : (
               <></>
              )}
          </GeoJSON>
```

```
        ))}
        {graph.myGeoJSON.features.map((feature, idx) => (
          <GeoJSON
            key={"lb- " + idx}
            data={feature}
            style={getStyle("black")}
          />
        ))}
        {graph.myGeoJSON.features.map((feature, idx) => (
          <GeoJSON
            key={"lf-" + idx}
            data={feature}
            style={getStyle(feature.properties.elevation)}
          />
        ))}
        {graph.myGeoJSONLable.map((feature, idx) => (
          <Marker position={feature.coordinates} key={"Marker
             -" + idx}>
            <Tooltip
              key={"Tooltip-" + idx}
              direction="right"
              offset={[-15, 20]}
              elevation={feature.elevation}
              className={colorTooltip(feature.event)}
              permanent={labelPermanent(feature.elevation,
                 takeZoomLvl())}
            >
              {`${feature.name} ${feature.event}`}
            </Tooltip>
          </Marker>
        ))}
      </Map>
    </>
  ) : (
    <NoData />
  )}
  </>
 );
};

export default MyMap;
```

Приложение 9

Компонента Table

Листинг кода реализации компоненты формирования таблиц реализуемых в приложении.

```
export const DataTable = ({ title, headers, rows, cellClass })
    => {
  const { classes } = useStoreState(state => state.classes);

  return (
    <>
      {title && <Title title={title} />}
      {headers.length > 0 && rows.length > 0 ? (
        <div className={classes.tableDiv}>
          <Table className={classes.table}>
            <TableHead>
              <TableRow>
                {headers.map((header, idx) => (
                  <TableCell
                    className={classes.tableHead}
                    key={`header-${idx}`}
                  >
                    {header.trim()}
                  </TableCell>
                ))}
              </TableRow>
            </TableHead>
            <TableBody>
              {rows.map((row, i) => (
                <TableRow
                  key={`row-${i}`}
                  style={{
                    backgroundColor: `${
                      row.hasOwnProperty("Event")
                        ? getColorOfEvent(row.Event)
                        : "white"
                    }`
                  }}
                >
                  {Object.values(row).map((column, j) => (
                    <TableCell
                      style={{
```

```
                        borderBottom: `${i === rows.length - 1
                            &&
                          "2px solid black"}`,
                        backgroundColor: `${column !== null &&
                          column.hasOwnProperty("color") &&
                          column.color}`
                      }}
                      className={cellClass ? cellClass :
                         classes.cell}
                      key={j}
                    >
                      {column === null
                        ? ""
                        : column.hasOwnProperty("percentage")
                        ? column.percentage
                        : column}
                    </TableCell>
                  ))}
                </TableRow>
              ))}
            </TableBody>
          </Table>
        </div>
      ) : (
        <NoData />
      )}
    </>
  );
};

export default DataTable;
```

Приложение 10

RESTful API для /api/start-report

Реализация RESTful API для конечной точки (endpoint) /api/ start-report

```
const express = require('express');
const router = express.Router();
const User = require('../../db/user');
const formidable = require('formidable');

router.get('/', function(req, res, next) {
  listM = [];
  User.teamList().then(list => {
    for(let i = 0; i < list.length; i++){
      listM.push(JSON.parse(list[i].file));
    }
    console.log(listM);
    if (!isNaN(req.query.id)) {
      User.getOne(req.query.id).then(user => {
        if (user) {
          //console.log(user);
          user = user.pop();
          //console.log(user);
          res.render('index', { title: 'Start report', menuLi:
              0, Udata: user, list: listM });
        } else {
          resError(res, 404, "User Not Found");
        }
      });
    } else {
      res.render('index', { title: ''Start report ', menuLi:
         0, Udata: false, list: listM });
    }
  });
});

router.post('/addfile*', function(req, res, next) {
  var form = new formidable.IncomingForm();
  form.parse(req, function (err, fields, files) {
    if (!isNaN(req.query.id)) {
      User.getOne(req.query.id).then(user => {
        if (user) {
          //console.log(user);
          user = user.pop();
```

```
            console.log(user, fields.team, JSON.stringify(fields
               .team));
            User.setteam(user, JSON.stringify(fields.team)).then
               (tt => {
              setTimeout(()=>{res.redirect("/");},150);
            });
          } else {
            resError(res, 404, "User Not Found");
          }
        });
      } else {
        setTimeout(()=>{res.redirect("/");},150);
      }
  });
});

function resError(res, statusCode, message) {
  res.status(statusCode);
  res.json({message});
}

module.exports = router;
```